SQL

I)Introduction :

C'est le passage d'une architecture centralisée à travers de grosse machine appelée mainframe vers une architecture distribuée basé sur une l'utilisation de serveur et de poste client grâce à l'utilisation des pc's et réseaux .

Cette évolution à été possible grâce à deux facteurs : Baisse des prix de l'informatique personnel, développement des réseaux.

II)L'architecture Client-Serveur

Définition

L'architecture client-serveur est un modèle de fonctionnement logiciel qui peut se réaliser sur tout types d'architecture matériel petites ou grosses machines à condition que les architectures soient inter-connectées ou puissent l'être. On parle de fonctionnement logiciel dans la mesure ou l'architecture est basée sur l'utilisation de deux types de logiciels : logiciels serveurs , logiciels clients . Qui s'exécute normalement sur des machines différentes .Dans cette architecture l'élément important est l'utilisation de mécanisme de communications entre deux applications . Le dialogue entre les applications peut se résumé par :

le client demande un service au serveur
le serveur réalise se service et renvoie le résultat au client.

INSERTION SCHEMA 1

Les principes généraux

Les principes qui régissent ce que l'ont entend par « client-serveur » sont :

Services : Le serveur est fournisseur de services et le client est consommateur .
Protocoles : C'est toujours le client qui déclenche la demande de service et le serveur attend passivement les requêtes des clients .
Partage des ressources : Un serveur traite plusieurs clients en même temps et contrôle leurs accès aux ressources.
Localisation : Le logiciel client-serveur masque au client la localisation du serveur.
Hétérogénéité : Le logiciel client-serveur est indépendant des plateformes matériels et logiciels.
Redimensionnement : Il est possible d'ajouter et de retirer des stations clientes et de faire évoluer les serveurs.
Intégrité : Les données du serveur sont géré de façon centralisé et les clients restent individuels et indépendant.
Souplesse & Adaptabilité : On peut modifier le module serveur sans toucher au module client et l'inverse également. Par exemple si une station est remplacée par un modèle plus récent on modifie le module client en améliorant l'interface , sans modifier le module serveur .

La répartition des tâches

Dans l'architecture client-serveur une application est constituée de trois parties :

L'interface utilisateur
La logique des traitements
La gestion des données

Le client n'exécute que l'interface utilisateur qui est souvent une interface graphique ainsi que la logique des traitements qui est formuler la requête pour laisser au serveur de Base de donnée la gestion complète des manipulations de données . La liaison entre le client et le serveur correspond à tout un ensemble complexe de logiciels appelés « middleware » qui se charge de toute les communications entre les processus .

4) Les différents modèle de clients-serveurs

Les différences sont essentiellement liées au service qui sont assurées par le serveur .

Le client-serveur de données

Le serveur assure des tâches de gestions , de stockages et de traitements de données .C'est le cas le plus connus de client-serveur , qui est utilisé par les plus grand SGBD. La base de donnée avec tout ses outils (maintenance , sauvegarde etc..) est installée sur un poste serveur . Sur les postes clients un logiciel d'accès est installé permettant d'accéder à la base de données du serveur . Tout les traitements sur les données sont effectuées sur le serveur qui renvoi les informations demandées par le client à l'aide d'une requête SQL.

Client-serveur de présentation

La présentation des pages affichées par le client est intégralement pris en charge par le serveur .L'inconvénient de cette organisation est de générer un fort trafic réseau .

Le client-serveur de traitement

Le serveur effectue des traitements à la demande du client , il peut s'agir de traitement particulier sur des données , de vérification de formulaire de saisie , de traitement d'alarme etc..
Ces traitements peuvent être réalisés par des programmes installés sur des serveurs mais également intégrés dans des base de données (triggers , procédures stockées etc..) Dans ce cas la partie donnée et traitement sont intégrés .

Synthèse des différents cas

INSERTION SCHEMA 2

1er cas) Présentation distribuée:

Correspond à l'habillage graphique de l'affichage en mode caractère d'application fonctionnant sur site central . Cette solution est aussi appelée revenping la classification du clien-serveur du revenping est souvent jugé abusive car l'intégralité des traitements originaux est conservé et que le poste client conserve une position d'esclave par rapport au serveur .

2ème cas) Présentation distante :

Elle s'appelle client-serveur de présentation , l'ensemble des traitements est exécutés par des serveurs et le client ne prend en charge que l'affichage . L'inconvénient de ce type d'application est de générer un fort trafic réseau et de ne permettre aucune répartition de la charge entre client et serveur.
Regain d'intérêt pour l'application avec l'exploitation des standard internet .

3ème cas) Gestion distante des données

Correspond au client-serveur de données c'est celui qui est le plus répandue . L'application fonctionne dans sa totalité sur le client , la gestion des données et le contrôle de leurs intégrités sont assurés par un SGBD centralisé. Cette architecture est souple et s'adapte très bien aux applications de types info-centre qui interroge la base de façon ponctuelle .Il génère toutefois un trafic réseaux assez important et ne soulage pas énormément le poste client qui réalise encore la grande majorité des traitements .

4ème cas) Traitements distribués

Correspond au client-serveur de traitement , le découpage de l'application se fait au plus près du noyau et les traitements sont distribués entre le clients et le(s) serveur(s). Le client-serveur de traitement s'appuie sur un mécanisme d'appel de procédures distantes ou sur la notion de procédures stockées proposé par les principaux SGBD.Cette architecture permet d'optimiser la répartition de la charge de traitement entre machine et limite le trafic du réseau.Par contre il n'offre pas la même souplesse que le client-serveur de données car les traitements doivent être connus du serveur à l'avance.

5ème cas) Base de données distribuée

Il s'agit d'une variante du client-serveur de données dans lequel une partie de donnée est prise en charge par le client .Ce modèle est intéressant si l'application doit gérer de gros volume de donner ou si l'ont souhaite disposer de temps d'accès très rapide sur des données pour répondre à de forte contrainte de confidentialité . Ce modèle est si puissant que complexe à mettre en œuvre .

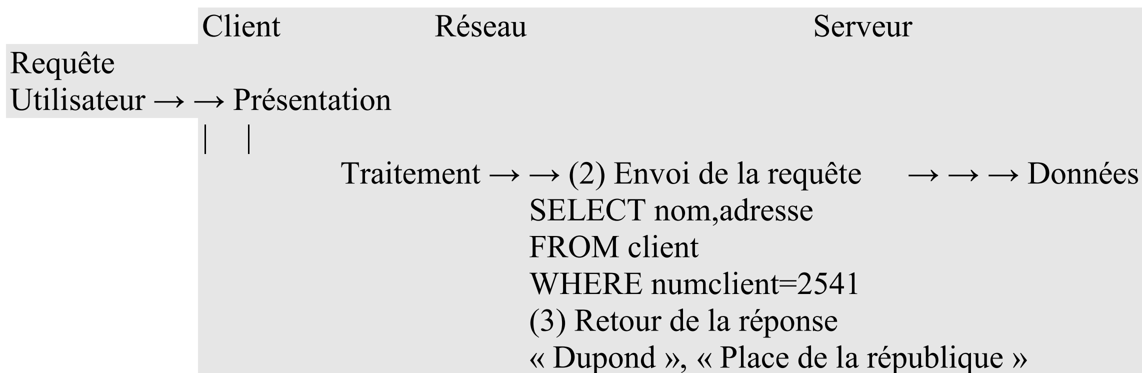
6ème cas) Données et traitements distribués

Ce modèle est très puissant et tire partie de notion de composant réutilisable et distribuable pour répartir au mieux la charge entre clients et serveurs . C'est bien entendue l'architecture la plus complexe.

5) Les différentes architectures

a) L'architecture 2 tiers

L'architecture 2/3 est appelé client-serveur de première génération ou client-serveur de données .Le poste client délègue la gestion des données à un service spécialisé . Par exemple : Une application fonctionnant sous WINDOWS ou UNIX et exploitant un SGBD centralisé . Ce type d'application permet de tiré partie de la puissance des ordinateurs déployé en réseau pour fournir à l'utilisateur une interface riche tout en garantissant la cohérence des données qui reste géré de façon centralisé. La gestion des données est prise en charge par un SGBD centralisé qui s'exécute souvent sur un serveur dédié. On l'interroge en utilisant un langage de requête comme le SQL. Le dialogue entre le client et le serveur se résume à l'envoie de requête et au retour de données correspond aux requêtes.



Cet échange de message transite à travers le réseau reliant les deux machines . Il met en œuvre des mécanismes complexe pris en charge par un middleware. L'expérience nous à montré qu'il était couteux et contraignant de vouloir faire porter l'ensemble des traitements applicatifs au client .On dit donc que le client est lourd ce qui donne un certain nombre d'inconvénient :

On ne peut pas soulager la charge du poste client .

Le poste client est fortement sollicité donc il est de plus en plus complexe et il nécessite des mise à jours régulière .

Les applications se prêtent assez mal au forte monté en charge car on ne peut modifier l'architecture initial .

La relation client-serveur complique les évolutions de cette organisation.

Ce type d'architecture est rigidifié par les coûts et la complexité de sa maintenance.

Les avantages :

L'utilisation d'une interface utilisateur riche .

Appropriation des applications par l'utilisateur.

Notion d'interopérabilité.

Pour résoudre les limitations du client-serveur 2/3 tout en conservant ses avantages. On a cherché une architecture plus évolué facilitant les forts déploiements à moindre coût. La solution est apporté par des architectures distribuées .

b) L'architecture 3 tiers

Les limites de l'architecture 2/3 provienne en grande partie des clients utilisés :

Le frontal complexe et non standard

Le middleware entre client et serveur n'est pas standard (SGBD , plateformes)

La solution est l'utilisation d'un poste simple communiquant avec le serveur par le biais d'un protocole standard .

L'architecture 3 tiers applique les principes suivants :

Données toujours agrées de façon centralisée.

Présentation toujours prise en charge pour le client.

Logique applicative prise en charge par un serveur intermédiaire.

Cette architecture 3/3 est appelé client-serveur de 2ème génération ou client-serveur distribué qui sépare l'application en 3 niveaux de service distinct et conforme aux principes suivant :

1er niveau : L'affichage + traitements locaux (contrôle de saisie , mise en forme de données …)

sont pris en charge par le client .
2éme niveau: Traitement applicatifs globaux pris en charge pour le service applicatif.
3ème niveau : Services de base de données pris en charge par un SGBD .

1er Niveau	2ème Niveau	3ème Niveau
Présentation → Traitements locaux	→ Traitements Globaux	→ Données

Tous ces niveaux sont indépendant donc ils peuvent être implanté sur des machines différentes ce qui permet :

Le poste client ne supporte plus l'ensemble des traitements , il est moins sollicité et peut être moins évolué donc moins coûteux
Les ressources présentent sur le réseau sont mieux exploitées car les traitements applicatifs peuvent être partagés ou regroupé .
La fiabilité et les performances de certain traitements sont améliorées par leurs centralisation
Il est relativement simple de faire face à une monté en charge en renforçant le service applicatif .

Dans cette architecture 3/3 le client est léger (thin client) . Le serveur de traitement constitue la pierre angulaire de l'architecture et ce trouve très sollicité .Dans ce type d'architecture il est difficile de répartir la charge entre client et serveur . On se retrouve confronté à un problème de dimensionnement serveur et gestion de la montée en charge comme à l'époque des mainframes.
Les solutions mise en œuvre sont complexe à maintenir et la gestion des sessions est compliquées .
Les contraintes semblent inversées par rapport à celles rencontrées avec les architectures 2/3 , le client est soulagé mais le serveur est fortement sollicité .

c) L'architecture « n » tiers

Requête SQL :

1) SELECT cli_nom, cli,pnom, ch_ville, cli_cp
FROM client
WHERE cli_cp LIKE 75
OR cli_cp LIKE '78%'
OR cli-cp LIKE '77%'
OR cli-cp LIKE '91%'
OR cli-cp LIKE '93%'
OR cli-cp LIKE '94%'
OR cli-cp LIKE '95%';

2
SELECT cli-nom, cli-pnom
FROM client
WHERE cli_mail is null

3
SELECT cli-nom, cli-pnom, c.cli_id, res_datadeb
FROM clientc, reservation r
WHERE c.cli-id=r.cli_id
AND res-date deb BETWEEN '07/01/2010' AND '07/31/2010';

4
SELECT cli_nom, cli_pnom, c.cli_id,e.emp_id

FROM emplacement
WHERE c.cli_id = r cli_id
AND e.emp_id = r emp_id
AND emp_ombre IS TRUE;

5
SELECT res _id
FROM reservation
WHERE res_id NOT in;
(SELECT res_id FROM lier;)

6
SELECT EMP_ID
FROM emplacement
WHERE em_elec is TRUE
AND emp-ombre is TRUE

7
SELECT e.emp_id
FROM emplacement, typempt
WHERE e.typ_id=typ_id
AND typ_lib LIKE 'vide'
SELECT e.emp_id

FROM EMPLACEMENT E, ZONE Z, ACTIVITE A

WHERE e.zon_id=z.zon_id

AND z.zon_id=a.zon_id

AND act_lib=PISCINE ;

9)

SELECT c.cli_id, r.resid

FROM reservation r, client c

Where c.cliud=r.cliud

And res_datefin <= 07/31/2010 ;

10)

SELECT E.EMP_id, r.res_datedeb, R.res_datefin

FROM emplacement E, Reservation R

Where E.emp_id=R.emp_id

And res_datedeb between 07/01/2010 and 07/31/2010

And res_datefin between 07/01/2010 and 07/31/2010

11)

```
SELECT  zon_lib, res_datedeb, res_date fin
FROM zone z, emplacement e, reservation r
Where z.zon_id = e.emp_id
And e.emp_id=r.emp_id
And r.res_detedeb between 07/01/2010 and 07/31/2010
And r.res_datefin between 07/01/2010 and 07/31/2010 ;
```

12)

```
SELECT c.cli_id, cli_nom, res_datedeb, res_datefin, (res_comptfin_comptdeb) as consummation
FROM CLIENT C, RESERVATION R
WHERE c.cli_id=R.cli_id
AND res_comptfin is not null;
```

13)

```
SELECT emp_id
FROM emplacement E, ZONE Z
WHERE z.zon_id=E.zon_id
AND z.zon_id not in;
```

14)

```
SELECT cli_nom, cli_id, m (reg-mnt) count (*)
FROM client c, reservation rs, reglement reg, lier l
WHERE c.cli_id=rs.cli_id
```

```
AND R.reg_id=L.reg_id

AND L.res_id=RS.res_id

GROUP BY c.cli_id,cli_nom

ORDER BY 3;
```

15)

```
SELECT sum (reg_mnt), count (*)

From reglement

Where reg_date    07/01/2010   ;
```

16)

```
SELECT SUM(reg_mat), count(*)

FROM REGLEMENT R, LIER L, RESERVATION RS

WHERE R.REG_id=L.reg_id

AND l.res_id=RS.res_id

AND res_datefin <=    07/31/2010   ;
```

2.2

```
INSERT INTO RESERVATION VALUES

(   RES_id   ,   EMP_id   ,    cli_id   ,    res_datedeb   ,
   res_datefin   ,   res_comptdeb   ,   res_comptfin   );
```

```
CREATE TABLE CLIENT

(   cli_nom   , VARCHAR 50 primary key, cli_pnom varchar 30

cli_adr VARCHAR 100,

cli-cp varchar 10,

cli_ville varchar  50,
```

```
cli_tel varchar 15,

cli_mail varchar 50) ;


CREATE TABLE EMPLACEMENT


(emp_id int 4 primary key,

zon_id int 4,

typ_id int 4,

emp_elec booleen

emp_ombre booleen

emp_point decimal,

emp_parking booleen);



insert into client values

(1,  dupont  ,  jenn  ,  5 rue montcey  ,  75009  ,  PARIS  ,  0123456789  , null) ;



Update tariff

Set tar_valeur= tar_valeur* 1,1

WHERE tar_datedeb >=  08/01/2010

AND tar_datefin <=  08/31/2010  ;
```

4)

```
UPDATE EMPLACEMENT

Set emp_point=emp_point+2

WHERE typ_id= (SELECT TYP_id FROM TYPEMP WHERE typ_lib=  mobile-home  );
```

5)

Insert into reservation(SELECT 20,10,cli_id, 07/10/2010 ,

07/22/2010 ,null,null

From client

WHERE cli_nom= dupont);

7)

Insert into activité (select 10, zon_id, terrain de petanque from zone

Where zon_lib= verte);

COURS SQL

TEE

Tee permet de créer un fichier texte avec toute les commandes saisis depuis la commande « tee » rentré .

Syntaxe :

```
tee nomfichier.txt
```

Pour stoper tee :

```
notee
```

ALTER TABLE

Permet la modification d'une table : Ajout d'un champ , modification d'un champ etc..

Syntaxe :
Permet la creation d'un champ de base à 4 Octets

```
ALTER TABLE SERVICE ADD numero INTEGER ;
```

```
ALTER TABLE SERVICE
ADD CONSTRAINT FK_numero FOREIGN KEY(numero)
REFERENCES zone(numero) ;
```

Script7.sql (exercice)

```
CREATE TABLE ZONE
( numero INT(1) PRIMARY KEY,
nom VARCHAR(20),
description VARCHAR(40)
) ;
ALTER TABLE SERVICE ADD numero INTEGER ;
```

```
ALTER TABLE SERVICE
ADD CONSTRAINT FK_numero FOREIGN KEY(numero)
REFERENCES zone(numero) ;

INSERT INTO zone
VALUES (1,"Nord","");
INSERT INTO zone
VALUES (2,"Est","");
INSERT INTO zone
VALUES (3,"Sud","");
INSERT INTO zone
VALUES (4,"Ouest","");

UPDATE SERVICE
SET numero=1
WHERE ville='Paris' ;
UPDATE SERVICE
SET numero=3
WHERE ville='Marseille' ;
```

Alias > Mettre des guillemets que quand il y a un espace dans l'alias

1) Afficher l'effectif de l'entreprise :

```
SELECT   COUNT(matricule)AS EFFECTIF
FROM     SALARIE      ;
```

2a) Afficher le nombre de salaries par service (N°Service)

```
SELECT   NSERVICE , COUNT(matricule)AS NBSAL
FROM            SALARIE
GROUP BY        NSERVICE   ;
```

2b) Idem en affichant en plus le nom du service presenter la liste dans l'ordre décroissant des effectifs .

```
SELECT   S.NSERVICE, NOMSERVICE, COUNT (matricule) AS NBSAL
FROM            SERVICE S,SALARIE Sa
WHERE    Sa.NSERVICE = S.NSERVICE
GROUP BY        S.NSERVICE,Sa,NSERVICE
ORDER BY        3 DESC   ;
```

"3" > On parle de la troisième "colonne" soit "COUNT(matricule) car on ne peut pas dire de trier via un alias et ceci évite de remettre "COUNT(matricule)" .
Jointure> syntaxe TABLE.CHAMP=AUTRETABLE.CHAMP

4) Afficher idem 3) pour les services qui ont au moins 3 salariés

```
GROUP
HAVING COUNT (matricule)>=3
ORDER BY
```

5) Afficher les matricules et noms des salaries qui gagnent plus que le salaire moyen

```
SELECT   matricule,nom,AVG(sal)
FROM            SALARIE
WHERE    Sal >
( SELECT        AVG(sal)
FROM            SALARIE ) ;
```

Autojointure
Afficher le nom de chaque salarié et le nom de son chef
Sql) SELECT salarie.nom , chef.nom FROM salarie , salarie chef WHERE salarie.nchef=chef.matricule ;

Inéquijointure
SELECT s.nom , s2.nom FROM salarie s , salarie s2 , service se WHERE s.nservice = se.nservice AND s2.nservice = se.nservice AND s.matricule < s2.matricue AND se.nomservice=”Etudes”;
SELECT s.nom , s2.nom, s.sal “salaire” FROM salarie s, salarie s2 WHERE s.sal=s2.sal AND s.matricule<s2.matricule ;